On Parallelization of the NIS-apriori Algorithm for Data Mining

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 60 (2015) 623 – 631

Procedia Computer Science

19th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

On Parallelization of the NIS-Apriori Algorithm for Data Mining

Mao Wu[a], Hiroshi Sakai[b,*]

[a]*Dwango Co. Ltd., Ginza, Tokyo, 104-0061, Japan*
[b]*Faculty of Engineering, Kyushu Institute of Technology, Tobata, Kitakyushu, 804-8550, Japan*

Abstract

We have been developing the *getRNIA* software tool for data mining under uncertain information. The *getRNIA* software tool is powered by the *NIS-Apriori* algorithm, which is a variation of the well-known *Apriori* algorithm. This paper considers the parallelization of the *NIS-Apriori* algorithm, and implements a part of this algorithm based on the *Apache-Spark* environment. We especially apply the implemented software to two data sets, the Mammographic data set and the Mushroom data set in order to show the property of the parallelization. Even though this parallelization was not so effective for the Mammographic data set, it was much more effective for the Mushroom data set.

Peer-review under responsibility of KES International

Keywords: data mining; *getRNIA*; *NIS-Apriori*; parallelization; rough sets; incomplete information; non-deterministic information;

1. Introduction

Rough set theory, proposed by Pawlak, gives us the mathematical framework for table data analysis[12,13,14]. This theory is applied to tables for mining rules, reading a tendency and a pattern, etc.[7,13,14]. In our study, we proposed the framework *Rough Non-deterministic Information Analysis* (*RNIA*), and push forward a study of the data mining technique in tables with non-deterministic information. We call such tables *Non-deterministic Information Systems* (*NISs*)[15,16,18].

In rough set theory, we usually handle tables with deterministic information, which we call *Deterministic Information Systems* (*DISs*). *NIS* and *Incomplete Information Systems* were proposed for dealing with information incompleteness in *DIS*[6,8,9,10,11]. Lipski employed the modal logic, and proved the logical properties in question-answering[9,10]. Orłowska investigated the certainty and the possibility in *NIS*[11]. We follow this robust framework, and we are developing the algorithms and the software tools in *RNIA*.

The *Apriori* algorithm is known well as the representative algorithm for data mining[1,2]. This algorithm deals with the *item* sets, which we call *transaction data*. For example, transaction data is automatically generated by using POS systems. However, if we identify an item with a descriptor [*attribute,attribute_value*] in *DIS*, we can similarly consider

* Corresponding author. Tel.: +81-93-884-3258 ; fax: +81-93-884-3258.
E-mail address: sakai@mns.kyutech.ac.jp

Peer-review under responsibility of KES International
doi:10.1016/j.procs.2015.08.198

the *Apriori* algorithm in *DIS*. We adjusted this *Apriori* algorithm in *DIS* to the *NIS-Apriori* algorithm in *NIS*. This is the core algorithm for our *getRNIA* system[17,19,20].

In this paper, we consider the parallelization of the *NIS-Apriori* algorithm for handling large scale data, and implement a part of this algorithm. Based on the experiment, the effectiveness of the parallelization was confirmed, especially for the Mushroom data set[5]. This paper is organized as follows: Section 2 recalls the rules in *RNIA*, and Section 3 reviews the *Apriori* algorithm and the *NIS-Apriori* algorithm as well as the *getRNIA* software tool. Section 4 investigates the parallelization of the *NIS-Apriori* Algorithm, and implements a part of this algorithm based on the *Apache-Spark* environment[4]. Finally, Section 5 concludes this paper.

2. Rules in Rough Non-deterministic Information Analysis (RNIA)

This section briefly surveys the framework of *RNIA*. A *Deterministic Information System DIS* ψ is a quadruplet below:[13,14]

$$\psi = (OB, AT, \{VAL_A \mid A \in AT\}, f), \quad f : OB \times AT \to \cup_{A \in AT} VAL_A, \tag{1}$$

where OB is a finite set whose elements are called *objects*, AT is a finite set whose elements are called *attributes*, VAL_A is a finite set whose elements are called *attribute values* and f is a mapping. We usually consider a table instead of this quadruplet ψ. *DIS* ψ_1 in Table 1 is an exemplary deterministic information system, and we see that the object x_1 means some implications, like $[Color, red] \Rightarrow [Weight, light]$ and $[Color, red] \wedge [Size, small] \Rightarrow [Weight, light]$.

Table 1. An exemplary *DIS* ψ_1.

Objects	*Color*	*Size*	*Weight*
x_1	*red*	*small*	*light*
x_2	*blue*	*small*	*light*
x_3	*red*	*small*	*heavy*
x_4	*blue*	*large*	*heavy*

Let us consider each implication τ below,

$$\begin{array}{l} \tau : \wedge_{A \in CON}[A, val_A] \Rightarrow [Dec, val], \ (val_A \in VAL_A, val \in VAL_{Dec}), \\ CON \subseteq AT : \text{(a set of) condition attributes}, \ Dec \in AT : \text{the decision attribute}. \end{array} \tag{2}$$

We say τ is a *rule* (or a *candidate* of a rule) in ψ, if τ satisfies a constraint in ψ. We say τ is *supported* by $x \in OB$ in ψ, if $f(x, A)$=val_A for every $A \in CON$ and $f(x, Dec)$=val hold. For specifying the object x, we may employ the notation τ^x. The most familiar constraint is defined by the following[14], and we also employ this constraint for two threshold values α and β $(0 < \alpha, \ \beta \leq 1.0)$.

$$\begin{array}{l} support(\tau^x) = |OBJ(\tau)|/|OB| \geq \alpha, \ accuracy(\tau^x) = |OBJ(\tau)|/|OBJ(\wedge_{A \in CON}[A, val_A])| \geq \beta. \\ \text{Here, } OBJ(*) \text{ means a set of objects supporting formula } *. \end{array} \tag{3}$$

NIS Φ is also a quadruplet below:[11,13,14]

$$\Phi = (OB, AT, \{VAL_A \mid A \in AT\}, g), \ g : OB \times AT \to P(\cup_{A \in AT} VAL_A) \text{ (a power set)}. \tag{4}$$

Every set $g(x, A)$ is interpreted as that there is an actual value in $g(x, A)$ but this value is not known[11,13,14]. By using *NIS*, it is possible to handle information incompleteness in *DIS*. Especially, if the actual value is not known at all, $g(x, A)$ is equal to VAL_A. This corresponds to the *missing value*[6]. We usually consider a table instead of this quadruplet Φ. Table 2 is an exemplary *NIS* Φ_2.

Now, we introduce the *derived DIS* from *NIS*. Since each VAL_A $(A \in AT)$ is finite, we can generate one ψ by replacing each non-deterministic information $g(x, A)$ with an element $v \in g(x, A)$. We named such ψ a *derived DIS* from *NIS*, and define the following:

$$DD(\Phi) = \{\psi \mid \psi \text{ is a derived } DIS \text{ from } NIS \ \Phi\}. \tag{5}$$

Table 2. An exemplary *NIS* Φ_2.

Objects	*Color*	*Size*	*Weight*
x_1	{*red*, *green*}	{*small*}	{*light*, *heavy*}
x_2	{*blue*}	{*small*, *medium*}	{*light*, *heavy*}
x_3	{*red*, *blue*}	{*small*, *medium*}	{*light*, *heavy*}
x_4	{*red*, *blue*}	{*large*}	{*heavy*}

In Φ_2, there are 256 (=2^8) derived *DISs*, and *DIS* ψ_1 is a derived *DIS* from Φ_2. Based on the interpretation of non-deterministic information, we see an actual *DIS* ψ^{actual} exists in 256 derived *DISs*. We consider the following two types of rules with modal concepts.

(*Certain rule*) An implication τ is a *certain rule*, if there is τ^x such that $support(\tau^x) \geq \alpha$ and $accuracy(\tau^x) \geq \beta$ in each $\psi \in DD(\Phi)$,
(*Possible rule*) An implication τ is a *possible rule*, if there is τ^x such that $support(\tau^x) \geq \alpha$ and $accuracy(\tau^x) \geq \beta$ in at least one $\psi \in DD(\Phi)$.

Remark 1. *In DIS* ψ, $support(\tau^x) = support(\tau^y)$ *and* $accuracy(\tau^x) = accuracy(\tau^y)$ *hold. So, we may identify* τ^x *with* τ. *However in NIS* Φ, *we may have such case that* τ^x *satisfies the constraint, but* τ^y *does not satisfy the constraint. If there is at least one* τ^x *satisfying the constraint, we see this* τ^x *is the evidence for the rule* τ.

We have $DD(\Phi) = \{\psi\}$ as the special case, and two types of rules define the same rules in ψ. Therefore, these two types of rules are the natural extension from rules in *DIS*. However, we need to pay attention to the number $|DD(\Phi)|$. In the Mammographic data set Φ_{Mammo} [5], $|DD(\Phi_{Mammo})|$ is more than 10 power 100.

3. Apriori Algorithm Adjusted to DIS, NIS-Apriori Algorithm, and the getRNIA Software

For adjusting the *Apriori* algorithm to *DIS*, we focus on descriptors and the structure of $\tau : \wedge_{A \in CON}[A, val_A] \Rightarrow [Dec, val]$. In the first step, we examine rules with one descriptor in the condition part of τ, i.e., rules in the form of $[A, val_A] \Rightarrow [Dec, val]$. In the second step, we examine rules with two descriptors, i.e., rules in the form of $([A, val_A] \wedge [B, val_B]) \Rightarrow [Dec, val]$. In the third step, we examine rules with three descriptors. Like this, we sequentially pick up any rule in the form of $\tau : \wedge_{A \in CON}[A, val_A] \Rightarrow [Dec, val]$. In this process, we employ the following properties:

(*The property on support*) $support(\tau) \leq \alpha$ holds, if $support(\wedge_{A \in CON'}[A, val_A]) \leq \alpha$ for at least one $CON' \subset CON$ or $support([Dec, val]) \leq \alpha$.
(*The property on accuracy*) $accuracy(\tau) \geq \beta$ may hold, even if $accuracy(\wedge_{A \in CON'}[A, val_A] \Rightarrow [Dec, val]) < \beta$ holds for every $CON' \subset CON$ ($CON' \neq CON$).

By employing these properties, we adjust each step in the *Apriori* algorithm to *DIS*.
(Step 1)
We generate CAN_1, CAN_{Dec}, and IMP_1 in the first step.

$$\begin{aligned} &CAN_1 = \{[A, val_A] \mid support([A, val_A]) \geq \alpha, A \in AT \setminus \{Dec\}\}, \\ &CAN_{Dec} = \{[Dec, val] \mid support([Dec, val]) \geq \alpha\}, \\ &IMP_1 = \{[A, val_A] \Rightarrow [Dec, val] \mid [A, val_A] \in CAN_1, [Dec, val] \in CAN_{Dec}\}. \end{aligned} \tag{6}$$

For each $\tau \in IMP_1$, we calculate $support(\tau)$ and $accuracy(\tau)$ for deciding whether τ is a rule or not. In this step, we recognize a set of rules in the form of $[A, val_A] \Rightarrow [Dec, val]$. We add this implication to $RULE_1$. For τ satisfying $support(\tau) \geq \alpha$ and $accuracy(\tau) < \beta$, we add this τ to $RESTR_1$.
(Step 2)
We generate IMP_2 in the second step. Because of the property on *accuracy*, we need to consider $RESTR_1$.

$$\begin{aligned} IMP_2 = \{&[A, val_A] \wedge [A', val_{A'}] \Rightarrow [Dec, val] \mid \\ &[A, val_A] \Rightarrow [Dec, val] \in RESTR_1,\ [A', val_{A'}] \Rightarrow [Dec, val] \in RESTR_1\}. \end{aligned} \tag{7}$$

For each $\tau \in IMP_2$, we calculate criterion values $support(\tau)$ and $accuracy(\tau)$ for deciding whether τ is a rule or not. In this step, we recognize a set of rules in the form of $[A, val_A] \wedge [A', val_{A'}] \Rightarrow [Dec, val]$. We add this implication to $RULE_2$. For τ satisfying $support(\tau) \geq \alpha$ and $accuracy(\tau) < \beta$, we similarly add this τ to $RESTT_2$. We sequentially continue this procedure until $IMP_n = \emptyset$. In each step, the properties on *support* and *accuracy* are effectively employed, and we recognize $\cup_i RULE_i$ is a set of all rules.

In the *Apriori* algorithm for the transaction data, the total search of the data set is employed frequently in order to calculate criterion values. In rough sets, we make use of the equivalence classes, and we always consider the equivalence class. Namely, we obtain a set $\{x \in OB \mid x \text{ supports } [A, val_A]\}$ for $[A, val_A]$ and a set $\{x \in OB \mid x \text{ supports } [Dec, val]\}$ for $[Dec, val]$ at the first step. By using the merging procedure[20], we are managing the equivalence class M for each implication τ, and we are calculating criterion values of τ $(=\tau^x)$ $(\forall x \in M)$. Since we are following rough set theory, we currently manage the equivalence classes, however we are also considering the total search of the data set instead of the equivalence classes.

Now, we cope with the *NIS-Apriori* algorithm. For considering this algorithm, we defined the following.

$$\begin{array}{l}
(1)\ minsupp(\tau^x) = min_{\psi \in DD(\Phi)}\{support(\tau^x) \text{ in } \psi\},\\
(2)\ minacc(\tau^x) = min_{\psi \in DD(\Phi)}\{accuracy(\tau^x) \text{ in } \psi\},\\
(3)\ maxsupp(\tau^x) = max_{\psi \in DD(\Phi)}\{support(\tau^x) \text{ in } \psi\},\\
(4)\ maxacc(\tau^x) = max_{\psi \in DD(\Phi)}\{accuracy(\tau^x) \text{ in } \psi\}.
\end{array} \tag{8}$$

We proved that it is possible to calculate the above criterion values in the polynomial time, and there is at least one $\psi_{min} \in DD(\Phi)$ causing the point $(minsupp(\tau^x), minacc(\tau^x))$. There is also at least one $\psi_{max} \in DD(\Phi)$ causing the point $(maxsupp(\tau^x), maxacc(\tau^x))$. The details are in the references 16 and 18. Based on these results, we obtained Figure 1 for each τ^x.

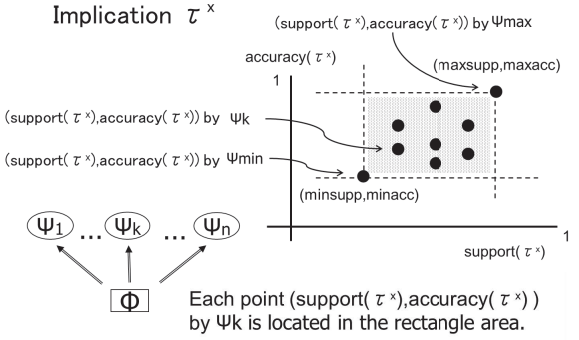

Fig. 1. Each pairs $(support(\tau^x), accuracy(\tau^x))$ in ψ $(\psi \in DD(\Phi))$ belongs to the rectangle area. In *DIS*, the minimum and the maximum points are the same, however they are different in *NIS*.

We have the following by using Figure 1.

(1) $support(\tau^x) \geq \alpha$ and $accuracy(\tau^x) \geq \beta$ hold for each $\psi \in DD(\Phi)$, if and only if $minsupp(\tau^x) \geq \alpha$ and $minacc(\tau^x) \geq \beta$.
(2) $support(\tau^x) \geq \alpha$ and $accuracy(\tau^x) \geq \beta$ hold for at least one $\psi \in DD(\Phi)$, if and only if $maxsupp(\tau^x) \geq \alpha$ and $maxacc(\tau^x) \geq \beta$.

Namely, we can handle certain rules by comparing $minsupp(\tau^x)$ and $minacc(\tau^x)$ with threshold values α and β, respectively. We can similarly handle possible rules by comparing the point $maxsupp(\tau^x)$ and $maxacc(\tau^x)$ with threshold values α and β. We adjusted *Apriori* algorithm in *DIS* to *NIS* by using the above properties. Since we can calculate criterion values in the polynomial time, the computational complexity of the *NIS-Apriori* algorithm is about the twice of the *Apriori* algorithm.

We opened a software *getRNIA* powered by the *NIS-Apriori* algorithm. In this web page, we can execute some demonstration files. In the Mammographic data set Φ_{Mammo}[5], $|DD(\Phi_{Mammo})|$ is more than 10 power 100, however we can easily obtained rules depending upon more than 10^{100} derived *DISs*.

4. Parallelization of the NIS-Apriori Algorithm and RNIA-Spark

This section reconsiders the parallelization of *NIS-Apriori*[21], and reports the current state of the implementation. The parallelization for data mining has been investigated by Agrawal[3]. In the reference 3, some parallelization processes based on the transaction data sets were considered.

4.1. Parallelization of NIS-Apriori

As we have shown in Section 3, *NIS-Apriori* generates the following sets sequentially.

$$\begin{aligned}
IMP_1 &= \{[A, val_A] \Rightarrow [Dec, val] \mid [A, val_A] \in CAN_1, [Dec, val] \in CAN_{Dec}\}.\\
IMP_2 &= \{[A, val_A] \wedge [A', val_{A'}] \Rightarrow [Dec, val] \mid \\
&\quad [A, val_A] \Rightarrow [Dec, val] \in RES T_1,\ [A', val_{A'}] \Rightarrow [Dec, val] \in RES T_1\}.\\
IMP_3 &= \{[A, val_A] \wedge [A', val_{A'}] \wedge [A'', val_{A''}] \Rightarrow [Dec, val] \mid \\
&\quad [A, val_A] \wedge [A', val_{A'}] \Rightarrow [Dec, val] \in RES T_2,\ [A, val_A] \wedge [A'', val_{A''}] \Rightarrow [Dec, val] \in RES T_2,\\
&\quad [A', val_{A'}] \wedge [A'', val_{A''}] \Rightarrow [Dec, val] \in RES T_2\}.\\
IMP_4 &= \quad : \quad : \quad :
\end{aligned} \tag{9}$$

For each implication $\tau \in IMP_k$ $(k = 1, 2, \cdots)$, we apply the calculation specified in Figure 1, and obtain the certain and possible rules. Figure 2 indicates this procedure.

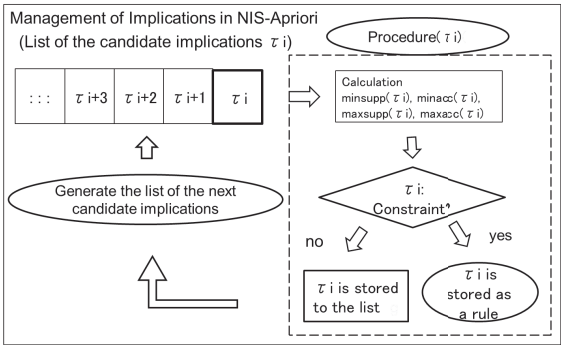

Fig. 2. The evaluation process of the implications by *NIS-Apriori*

We focused on this procedure, and considered the parallelization in Figure 3. In Figure 2, each implication τ_i is examined sequentially. In Figure 3, the list of implications are divided into four sub-lists, and each list is handled simultaneously. We implemented a software tool for the fixed number of the core processors[21], and newly revised this software tool so as to detect the number of the core processors automatically. The new software tool generates the sub-lists based on the number of the core processors, and assigns each procedure to each core processor. Even though this is more general software than the previous implementation, the implementation is restricted to the set IMP_1. As for the set IMP_2 and IMP_3, we are still in progress.

In the following, we know spark1p.py takes one core processor, and we see each task is executed sequentially. On the other hand, spark_multi.py takes four core processors, and we see each task is executed at the same time.

```
>pyspark ../rnia_spark/rnia_spark1p.py local data/mush3.pl
15/02/10 09:04:05 INFO Executor: Running task ID 1
15/02/10 09:04:08 INFO Executor: Running task ID 2
15/02/10 09:04:11 INFO Executor: Running task ID 3
15/02/10 09:04:13 INFO Executor: Running task ID 4
```

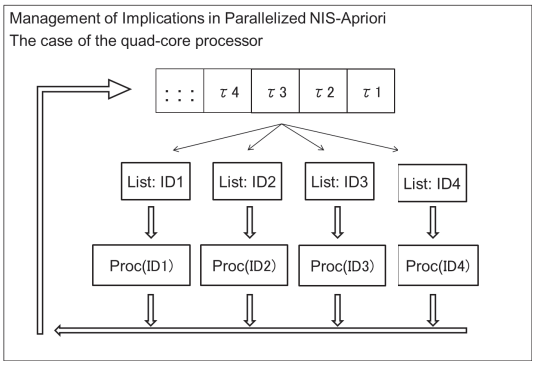

Fig. 3. The evaluation process of the implications for the quad-core processor.

```
>pyspark ../rnia_spark/rnia_spark_multi.py local data/mush3.p
15/02/10 09:07:09 INFO Executor: Running task ID 1
15/02/10 09:07:09 INFO Executor: Running task ID 2
15/02/10 09:07:09 INFO Executor: Running task ID 4
15/02/10 09:07:09 INFO Executor: Running task ID 3
```

4.2. Apache-Spark Environment

Spark is a MapReduce-like data-parallel computation engine open-sourced by UC Berkeley. The Spark Python API (*PyS park*) exposes the Spark programming model to Python. At a high level, every *Spark* application consists of a driver program that runs the user's *main* function and executes various parallel operations on a cluster. We employ this environment for implementing the software tools for *RNIA*, and we call this framework *RNIA-Spark*. In *RNIA-Spark*, we distribute rule generation operations of different descriptors to a cluster which may consist of multiple processors.

The main abstraction Spark provides is a resilient distributed dataset (*RDD*), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. In *RNIA-Spark*, we try to partition the whole rule generation task to separate rule generation tasks of difference descriptors.

4.3. An Automated Detection of Core Processors

However in the previous version of *RNIA-Spark*, we hard-coded the number of processors for parallelized rule generations. In this paper we use Python's multiprocessing module to check system specs, and optimize the parallel execution based on the number of cores.

Firstly, we need to import the *multiprocessing* module and call the *cpu_count* function to obtain the number of cores, and assign it to a variable *NCore*, which will be used in the main function later.

```
# check number of cpus >python2.6
import multiprocessing
NCore = multiprocessing.cpu_count()
```

Then, we need to import *SparkContext* and define the *main* function, which accepts several arguments needed when we run *RNIA-Spark* from command line afterward. The first thing a *Spark* program must do is to create a *S parkContext* object, which tells *Spark* how to access a cluster. For example, the first parameter *sys.argv*[1] is a string specifying a *Spark* or *Mesos* cluster URL to connect to, or a special *local[NCore]* string to run in local mode.

The second parameter *rnia_spark* is the application name, which will be shown in the cluster web UI. The *plCleaning* here is the data cleaning function used to convert raw $*.pl$ files or *csv* files to formatted data. The following *ruleGeneration* function is the core of *RNIA-Spark*, which applies *NIS-Apriori* algorithm on formatted data based on the settings we configured in the raw file. It returns a list of rules or an empty list if none of the rules satisfy. Finally, the $dTCF2list$ function's job is to convert the list of results to readable texts.

```
from pyspark import SparkContext
def ruleGeneration(argW, orDsp=None):
    ...
    dctps=sc.parallelize(
        list(itertools.product(
            xrange(0,len(AT[D])),
            condition_Indexes)),
             NCore)
    ...

if __name__ == "__main__":
    if len(sys.argv) < 3 or len(sys.argv)>4:
        print >> sys.stderr, "Usage: ./pyspark  rnia_spark_multi.py <master> <file>"
        exit(-1)
    localCore="local[{NCore}]".format(NCore=NCore)
    sc = SparkContext(localCore, "rnia_spark")
    argWrapper=plCleaning(file_name)
    dataWrapper= ruleGeneration(argWrapper)
    rules=_dTCF2list(dataWrapper)
```

4.4. An Implementation and Experiments

To take full advantage of rnia_spark_multi.py, we need a computer of more than 4 cores or a cluster of more than 4 nodes. We obtain the following comparison result on an 8-core PC.

Firstly, we execute rnia_spark_multi.py on data Mammo.pl, which has 960 objects and 6 attributes per object. The Mammo.pl is the revised Mammographic data set[5] as we have specified in Section 2. The result proves that the *rnia_spark_multi* version is not very efficient because the cost of parallel procedure itself is much more expensive than the efficiency it brings on such scale of data.

```
>pyspark rnia_spark/rnia_spark1p.py local data/mammo.pl
Number of cores: 1
 Data Cleaning time: 0.315397977829
 Rule Generation time: 0.308684110641

>pyspark rnia_spark/rnia_spark4p.py local data/mammo.pl
Number of cores: 4
 Data Cleaning time: 0.312597036362
 Rule Generation time: 0.241578102112

>pyspark rnia_spark/rnia_spark_multi.py local data/mammo.pl
Number of cores: 8
 Data Cleaning time: 0.352070093155
 Rule Generation time: 0.268180847168
```

The following figure is the execution time (the difference between the finishing time and the starting time) of the duplicated Mushroom data set[5] including descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Every Mushroom data set has 8124 objects and 22 attributes per object which may contain non-deterministic attribute values. From top to bottom, the lines are the result of *RNIA-Spark* with single processor , result with 4 processors, and result with 8 processors. Obviously, *RNIA-Spark* on multiple processors become more efficient when the dataset grows bigger.

As for this parallelization, we can easily have the following.
(1) The parallelization will be effective, if IMP_k has the large number of implications.
(2) The parallelization may not be effective, if IMP_k has the small number of implications.
(3) Since IMP_k depends upon the threshold values α and β, the parallelization will be effective for α and β with lower values. The parallelization may not be effective for α and β with higher values.
(4) The rule generation from the Mushroom data set will correspond to the effective case, and the rule generation from the Mammo.pl data set will correspond to the ineffective case.

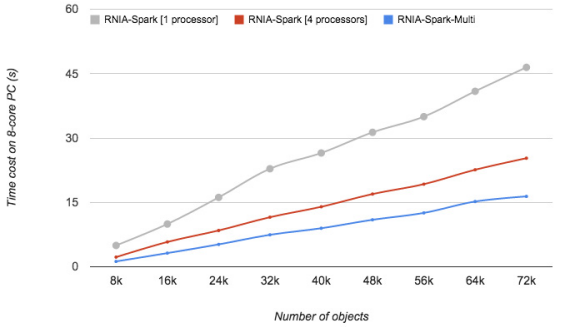

Fig. 4. Execution time for the Mushroom data set by the multiple core version of *RNIA-Spark*

5. Concluding Remarks

This paper briefly surveyed rough sets in *DIS*, rough sets in *NIS*, *RNIA* and rule generation. We implemented the web software *getRNIA* based on the *NIS-Apriori* algorithm, and opened it to the public [17,19,20]. This is implemented in Python, and employs Google App Engine. We are now adding the parallelization functionality to the *NIS-Apriori* algorithm, especially the parallelization for the evaluation of the criterion values by using *Apache-Spark* environment. As Agrawal described[3], we need to consider the parallelization in other procedures. Even though our work is in progress, we think the parallelization of the algorithm will take the important role for analyzing big data.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work is supported by JSPS (Japan Society for the Promotion of Science) KAKENHI Grant Number 26330277.

References

1. Agrawal, R., Srikant, R. Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C., editors. *VLDB*, Morgan Kaufmann; 1994. p. 487–499.
2. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I. Fast discovery of association rules. In: Usama M. Fayyad, et al., editors. *Advances in Knowledge Discovery and Data Mining* AAAI/MIT Press; 1996. p. 307–328.
3. Agrawal, R., Shafer, C. Parallel mining of association rules. *IEEE Transaction on Knowledge and Data Engineering* 1996;**8**(6):962–969.
4. Apache Spark project Homepage
`https://spark.apache.org/`
5. Frank, A., Asuncion, A. UCI Machine Learning Repository, Irvine, CA, 2010.
`http://mlearn.ics.uci.edu/MLRepository.html`
6. Grzymała-Busse, J.W., Werbrouck, P. On the best search method in the lem1 and lem2 algorithms. Incomplete Information: Rough Set Analysis *Studies in Fuzziness and Soft Computing* 1998;**13**:75–91.
7. Komorowski, J., Pawlak, Z., Polkowski, L., Skowron, A. Rough sets: a tutorial. *Rough Fuzzy Hybridization: A New Method for Decision Making* Springer; 1999. p. 3–98.
8. Kryszkiewicz, M. Rough set approach to incomplete information systems. *Information Sciences* 1998;**112**(1-4):39–49.
9. Lipski, W. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems* 1979;**4**(3):262–296.
10. Lipski, W. On databases with incomplete information. *Journal of the ACM* 1981;**28**(1):41–70.
11. Orłowska, E., Pawlak, Z. Representation of nondeterministic information. *Theoretical Computer Science* 1984;**29**(1-2):27–39.
12. Pawlak, Z. Rough sets. *International Journal of Computer and Information Sciences* 1982;**11**:341–356.
13. Pawlak, Z. *Systemy Informacyjne: Podstawy teoretyczne*. Wydawnictwa Naukowo-Techniczne Publishers; 1983.
14. Pawlak, Z. *Rough Sets: Theoretical Aspects of Reasoning About Data*. Kluwer Academic Publishers; 1991.

15. Sakai, H., Okuma, A. Basic algorithms and tools for rough non-deterministic information analysis. *Transactions on Rough Sets* 2004;**1**:209–231.
16. Sakai, H., Ishibashi, R., Koba, K., Nakata, M. Rules and apriori algorithm in non-deterministic information systems. *Transactions on Rough Sets* 2008;**9**:328–350.
17. Sakai, H. *RNIA software logs*, 2011.
`http://www.mns.kyutech.ac.jp/~sakai/RNIA`
18. Sakai, H., Wu, M., Nakata, M. Apriori-based rule generation in incomplete information databases and non-deterministic information systems. *Fundamenta Informaticae* 2014;**130**(3):343–376.
19. Wu, M., Sakai, H. *getRNIA web software*, 2013.
`http://getrnia.org/`
20. Wu, M., Nakata, M., Sakai, H. An overview of the *getRNIA* system for non-deterministic data. *Procedia Computer Science* 2013;**22**:615–622.
21. Wu, M., Yamaguchi, N., Liu, C., Sakai, H. Toward the enhancement of the *getRNIA* system for rough-set based data analysis. *Proc. SCIS-ISIS2014* 2014; TP5-2-5-(3).